单调优化

2019年3月14日 14:00

-
- 单调栈

一. 单调优化思想
   1. 保存具有单调性的数据，及时排除无关数据，保持策略集合高度有效和秩序性，为做出决策提供更多条件和方法
   2. 单调栈：内部数据按进栈顺序有单调性，若新数据不满足单调性，选择不放入该数据或弹出部分旧数据
   3. 单调双端队列：内部数据按进队顺序有单调性，除了可以类似单调栈地弹出队尾外，还可类似滑动窗口地排除陈旧的队头

二. 保留所有字符的最小字符串（使原串每个字符出且仅出现一次，且字典序最小）
   1. 引：能输出单增解的最优情况：用单调栈保存筛选后的单增串，如果遍历到的新字符恰比栈顶大，直接入栈，如果比栈顶小，弹到比栈顶大为止
   2. 转：如果有的较大字符出现次数很少，使被弹出后可能就不会再出现，显然不符题意，可以记录字符出现的最后位置，确保其最后出现位置在当前遍历位置之后才弹出该较大字符
   3. 转：为了让字符只出现一次，可以开一个遍历标记v
   4. 转：为了方便输出，用deque代替stack
   5.
```
string s;
map<char,int>pos;
map<char,bool>vq;
deque<char>q;

int main(){
	cin>>s;
	int len=s.size();
	for_(i,0,len)
		pos[s[i]]=i;	//更新s[i]最后一次出现的下标
	for_(i,0,len)
		if(!vq[s[i]]){	//得想办法插入这个字母
			while(!q.empty()&& q.back()>s[i]&& pos[q.back()]>i){
				vq[q.back()]=false;
				q.pop_back();}
			q.push_back(s[i]);
			vq[s[i]]=true;}
	cout<<string(q.begin(),q.end());
	return 0;}
```

三. 柱状图中的最大矩形
   1. 即从若干并排的不同高矩形里切割出一个完整矩形，求其最大面积
   2. 引：矩形高度递增时，从左至右以每个矩形的高度乘以后缀矩形宽度和，遍历到的可能值的最大值即为答案
   3. 转：当出现一块矩形比左边矩形高度小，则左边比它高的所有矩形都不能运用引言的方法，可以以这个矩形为终点，计算每个比当前矩形高的左矩形可能取到哪些

值，之后就直接忽视左边这些高矩形

4. 结：从左往右遍历，用一个栈保存从左往右高度单增的矩形高度，若新的数据比栈顶数据低，则循环弹出数据，一边弹一边用弹出的累积宽度乘以当前弹出矩形的高度，并尝试更新数据，弹到栈顶比新遍历到的数据低或栈空为止，把累积弹出的宽度记录在新栈顶
5. 为了方便处理最后栈内剩余的数据，可以事先在高度数组末加一个高度为0的数据
6.
```
a[n+1]=s[0]=p=w[0]=0;
//矩形高度，栈，栈顶指针，栈内数据的宽度
for__(i,1,n+1){
	if(a[i]>s[p])	//直接入栈
		s[++p]=a[i],
		w[p]=1;
	else{
		int width=0;
		while(s[p]>a[i]){
			width+=w[p];
			ans=max(ans,(ULL)s[p]*width);
			--p;
		}
		s[++p]=a[i];	//入栈
		w[p]=width+1; //将弹出矩形视作与它一样高的矩形
	}
}
```

四. 最长j-i

1. 在下标i~j的数据中，a[i]是最小值，a[j]是最大值，求这种j-i的最大值
2. 引：如果i是最小值下标，那么之后每次找到i后区间最大值，就更新一次j-i
3. 转：如果发现a[j]不仅不是最大值，还比a[i]小，就把这个j视作新i
4.
```
int si[1005],sd[1005]; //单增栈，单减栈
int pi,pd;	//单增栈顶下标，单减栈顶下标
int a[1005],ans;
int n;
sd(n);
for_(i,0,n)
	sd(a[i]);
sd[0]=si[0]=ans=pi=pd=0;
for_(j,1,n){
	while(pd>=0 && a[j]>a[sd[pd]])
		--pd;	//弹出单减栈中所有比新数据小的数据
	sd[++pd]=j;	//插入新数据
	if(pd==0)	//当前a[j]是目前子区间的最大数据
		ans=max(ans,j-si[0]);
	//单增栈si[0]是目前子串最小数据的下标，即题目里的i
	while(pi>=0 && a[j]<a[si[pi]])
		--pi;	//弹出单增栈中所有比新数据大的数据
	si[++pi]=j; //插入新数据
	if(pi==0)	//当前a[j]是目前子区间的最小数据
		pd=0,	//不用管之前的大数据了，新开一个单减栈
		sd[0]=j;
}//为了防止新开单减栈影响到更新ans，所以后维护单增栈
printf("%d\n",ans);
```

-
- 单调队列

一. 长度不超过k的各个区间里的最大值，及其序号（越左越好）

1. 引：用队列q维护递减序列的序号，则最左的序号对应数据即为答案

2. 转：区间长度不超过k，因此发现队列首尾长度>=k就扔掉头
3. 结：while n尾>ni ： 弹尾；if 尾-头>=k ： 弹头；
4.
```
for__(i,1,n)
        scanf("%d",a+i);
int l= 1;
int r= 0;
q[++r]= 1;
for_(i,2,k){
        while(l<=r && a[q[r]] < a[i])
                --r;
        q[++r]= i;}
for__(i,k,n){
        while(l<=r && a[q[r]] < a[i])
                --r;
        q[++r]= i;
        if(q[r]-q[l] >= k) //不是长度为k的区间
                ++l;
        printf("%d %d\n",a[q[l]],q[l]);}
```

二. 长度恰为m的各区间的Max及更新Max的次数（USSTD2H）

1. 题：求Σ Max 异或 区间左端点 和 Σ 更新次数 异或 区间左端点
2. 引：前半部分即为上一题维护长度恰为M的单调递减队列，插入完下标i以后，i-M+1即为当前队列对应区间的左端点
3. 转：反着跑滑动窗口，插入完第i个下标的后，单减队列中存的每个下标对应的数据都比当前i大（因为是反向遍历的），也就是正向跑时更新答案的次数，而i本身即为左端点
4. 结：正反各跑一遍维护单减滑动队列，当i对应的区间满足长度>=m时更新

```
int l=0; //队首下标
int r=0; //队尾的后一个下标
for__(i,1,n){
        if(/*l<r && */q[l]<=i-m)
                ++l;      //长度爆了，该换队头了
        while(l<r && a[q[r-1]]<=a[i])
                --r;      //新队尾巨大，可以扔掉很多无用的队尾
        q[r++]=i;          //先存入新队尾再更新队尾指针
        if(i >= m)
                Mrt+= a[q[l]]^(i-m+1);}
l= r= 0;
rof__(i,n,1){
        if(/*l<r && */q[l]>=i+m)
                ++l;      //长度爆了，该换队头了
        while(l<r && a[q[r-1]]<=a[i])
                --r;      //新队尾巨大，可以扔掉很多无用的队尾
        q[r++]=i;          //先存入新队尾再更新队尾指针
        if(n-i+1 >= m)
                cnt+= (r-l)^i;}
```

三. 长度不超过m的最大子区间和

1. 引：下标递增且值也递增的前缀和s[i]-s[j]中的最大值即为答案
2. 转：下标差超过m的数据s[i]应忽视；不递增的前缀和应忽视

3. 结：用队列保存递增的前缀和的下标，队首即为适合被减的前缀和下标
4. 注：ans初值推荐是负无穷-0x3f3f3f3f
5.
```
int l=0,r=0;
q[0]=0;
for__(i,1,n){
	if(i-q[l]<m)
		++l;	//弹出队首
	ans=max(ans,s[i]-s[q[l]]);
	while(l<=r && s[q[r]]>=s[i])
		--r;	//弹出队尾
	q[++r]=i;	//更新队尾
}
```

四. 直线上跳格游戏（USSTD4I）

1. 题：已知n个坐标在x，价值为s的特殊点，目标获得k分后结束游戏。起始每次恰能跳d格，可以花费g元让可跳步数范围左右区间各增减g步，求通关最少g
2. 引：g越大选择越多，答案有单调性，二分查即可
3.
```
bool ok(ll g){
	ms(dp,0xc0);	//初值负无穷
	dp[0] = 0;	//起点
	int cur = 0;	//当前所在位置
	ll up= d+g;	//步伐上限
	ll dn= max(1ll,d-g);	//步伐下限
	for__(i,1,n){	//遍历每个打算跳的点i
		while(l<r && x[q[l]] < x[i]-up)
			++l;	//清掉跳不到i的队头
		while(cur<=n && x[cur] < x[i]-up)
			++cur;	//找到第一个跳得到i的cur
		while(cur<=n && x[cur] <= x[i]-dn){	//遍历每个跳得到i的cur
			while(l<r && dp[q[r-1]] <= dp[cur])
				--r;	//清掉不如cur值钱的队尾
			q[r++] =cur;	//把跳得到i的cur入队
			++cur;
		}	//现在队列里都是跳得到i的位置，且值钱程度递减
		if(l<r)	//非空
			dp[i]= s[i] + dp[q[l]];
		if(dp[i]>=k)
			return 1;	//可以结束游戏
	}
	return 0; }
```
4.
```
while(l<r){
	ll g= l+r >>1;
	if(ok(g))
		r=g;
	else
		l=g+1;
	//cout<<g<<"  "; for__(i,0,n) cout<<dp[i]<<" "; cout<<'\n';
}
printf("%lld", ok(l)? l: -1);
```

五. 和恰为t的区间最短长度

1. 遍历每个区间右端点r，仅当区间和超过t时弹出左端点
2. ```
   for_(i,0,n)
       cin>>a[i];
   if(a[0] == t){
       cout<<1;
       return 0;}
   int l= 0;
   ll sum= a[0];
   for_(r,1,n){
       sum+= a[r];
       while(sum>t)
           sum-= a[l++];
       if(sum==t)
           ans= min(ans, r-l+1);}
   if(ans <= n)
       cout<<ans;
   else
       cout<<"No";
   ```
3. 顺带一提利用STL二分搜索前缀和的遍历左端点的O（logN）算法如下
   ```
   for_(i,0,n)
       cin>>a,
       s[i+1] = s[i]+ a;
   for__(i,0,n)
       if(*lower_bound(s+i+1, s+n+1, t+s[i]) == t+s[i])
           ans= min(ans, int(lower_bound(s+i+1, s+n+1, t+s[i]) - s - i));
   ```
   ```